Lab_04_LogisticReg_NonLin.m

```
%% Machine Learning
% Lab 4: Regularized Logistic Regression (NonLinear case)
% ---- Microchip Anomaly ----
%{
In this part of the exercise, you will get to try out different regularization
parameters for the dataset to understand how regularization prevents overfitting.
Notice the changes in the decision boundary as you vary lambda. With a small
lambda, you should and that the classifier gets almost every training example
correct, but draws a very complicated boundary, thus overfitting the data.
This is not a good decision boundary: for example, it predicts
that a point at x = (-0.25; 1.5) is accepted (y = 1), which seems to be an
incorrect decision given the training set.
With a larger lambda, you should see a plot that shows an simpler decision
boundary which still separates the positives and negatives fairly well. How-
ever, if lambda is set to too high a value, you will not get a good at and the decision
boundary will not follow the data so well, thus underfitting the data
%}
%{
%  In this part, you are given a dataset with data points that are not
%  linearly separable. However, you would still like to use logistic
%  regression to classify the data points.
%
%  To do so, you introduce more features to use -- in particular, you add
%  polynomial features to our data matrix (similar to polynomial
%  regression).
%}

%% Initialization
clear ; close all; clc

%% Load Data
%  The first two columns contains the X values and the third column
%  contains the label (y).

data = load('ex2data2.txt');
X = data(:, [1, 2]); Y = data(:, 3);

plotData(X, Y);

% Put some labels
hold on;
% Labels and Legend
xlabel('Microchip Test 1')
ylabel('Microchip Test 2')
% Specified in plot order
legend('y = 1', 'y = 0') % Good or Nay
hold off;

%% Add Polynomial Features

% Note that mapFeature also adds a column of ones for us, so the intercept
% term is handled
X = mapFeature(X(:,1), X(:,2));

% Initialize fitting parameters
```

```
init_w = zeros(size(X, 2), 1);

% Set regularization parameter lambda to 1 (you should vary this)
%  Try the following values of lambda (0, 1, 10, 100).
lambda = 0.1;

%% Set Options
options = optimset('GradObj', 'on', 'MaxIter', 400);

% Optimize
[w, J, exit_flag] = ...
        fminunc(@(t)(costFunctionReg(t, X, Y, lambda)), init_w, options);

% Plot Boundary
plotDecisionBoundary(w, X, Y);
hold on;
title(sprintf('lambda = %g', lambda))

% Labels and Legend
xlabel('Microchip Test 1')
ylabel('Microchip Test 2')

legend('y = 1', 'y = 0', 'Decision boundary')
hold off;

% Compute accuracy on our training set
p = predict(w, X);

fprintf('Train Accuracy: %f\n', mean(double(p == Y)) * 100);
fprintf('Expected accuracy (with lambda = 1): 83.1 (approx)\n');
```

mapFeature.m

```
function out = mapFeature(X1, X2)
% MAPFEATURE Feature mapping function to polynomial features

degree=3; % change it to 6 for example
out = ones(size(X1(:,1))); % first column ones -> bias
for i = 1:degree
    for j = 0:i
        out(:, end+1) = (X1.^(i-j)).*(X2.^j);
    end
end

end
```

sigmoid.m

```
function g = sigmoid(z)
%SIGMOID Compute sigmoid function
g = zeros(size(z));
g = 1 ./ (1 + exp(-z));
end
```

costFunctionReg.m

```
function [C, grad] = costFunctionReg(w, X, Y, lambda)
%COSTFUNCTIONREG Compute cost and gradient
%for logistic regression with regularization

m = length(Y); % number of training examples
C = 0;
grad = zeros(size(w));

% calculate cost function
h = sigmoid(X*w);
% calculate penalty
% excluded the first w value
w1 = [0 ; w(2:size(w), :)];
p = lambda*(w1'*w1)/(2*m);
% Alternative:
%p = (lambda / (2*m))*sum(w_reg.^2);

C = ((-Y)'*log(h) - (1-Y)'*log(1-h))/m + p;
% Alternative:
% C = ((1/m).*sum(((-1.*Y).*log(hx)) - ((1.-Y).*log(1.-hx))))+p;

% calculate grads
grad = (X'*(h - Y)+lambda*w1)/m;

end
```